Research Needs in Verification
Semiconductor Research Corporation
March 2002

This document describes needs and problems SRC member companies see in the verification of hardware and system designs. It is not intended to restrict research proposals or to discourage innovative thinking, but to point potential university investigators to the challenges members face and to encourage work in these key areas. Successful proposals in response to these needs will advance the state of the art, open new opportunities to SRC members, and provide new ways to assure error-free products. Proposals for new description languages and their compilers, or for incremental improvements to existing techniques, are not encouraged.

Tools for Architectural and High Level Abstraction for Hardware Design

– What are formal ways of relating high level language functional models to low level structural descriptions?
– How can a "golden model" for a process or system specification be defined and used?
– Can declarative models be used to describe and evaluate hardware components or systems on a higher level of abstraction?

Example: Designers often start with an architectural model written in C or a similar high-level, functional, executable language with annotations, which is extensively simulated and revised, until it becomes a "golden standard". This standard is used to guide circuit implementation in RTL, which can then be "co-simulated" with the golden model to look for discrepancies. Since the C model is not cycle-accurate, correspondence points must be designated manually. The co-simulation is also inefficient, and a sense of sufficient "coverage" is hard to come by. A possible solution would be to extract a finite-state model from the C code, and compare it with the RTL model using formal methods. There are serious challenges in reduction to finite state models, development of equivalence relations, etc.

Verification of Processors and Subsystems

– Is there an automatic way of verifying arithmetic and floating point operations?
– How can protocols, memory arrays, and deeply pipelined machines best be modeled and verified?
– What properties in addition to function (performance, area, power, I/O behavior, timing) can be verified, and how?
– How can constraint satisfaction techniques be used for generation of test vectors for verification of floating point units, and for test-program generation for high-end designs?

Example: Development of CSP solutions to cope with verification of FP units, including units compliant with the IEEE standard but also units that extend the standard (e.g., special format, FP multiply-add instruction, etc.) or slightly deviate from it.

Example: Development of CSP solutions to cope with the specific attributes posed by constraint networks introduced during test-program generation, namely: (1) random well-distributed solutions (2) Huge variable domains (e.g., processor's address space) (3) Combination of linear constraints with non-linear non-monotonic constraints (e.g., bit-wise XOR)

System-on-Chip and System-Level Verification (including Mixed Signal)

– How can properties be successfully specified and verified at the system level?
– How can "verified" heterogeneous components be assembled into a hardware system so that properties can be proved about the whole system?
– Are there cost-effective/practical tools for verifying microcode and microarchirectures?
– What are the approaches for verifying heterogeneous components and their combinations, including mixed signal/analog?

Example: Verification of analog, RF, mixed signal, MEMS, and other components would be a great advance for formal methods, but is extremely difficult – analog test is still a challenge.

Example: SRC remains focused on hardware verification, but system verification will of necessity involve software components, particularly as applied to embedded kernels, firmware, auto-generated code, etc.

Verification Methods in the Design Flow

– How can bugs in design be reduced by making designs easier to verify and bugs easier to detect above the RTL level?
– What verification methodologies exist for *practical* verification?
– What formal techniques can be used in the design flow to aid verifiability?
– What combined methods with increased capacity will allow more fast "yes" and "no" answers to large problems to detect more bugs earlier?
– How can an environment from a component be automatically generated for assume/guarantee methods?
– How can formal and mathematical methods be used to derive patterns for simulation and the sequencing of test cases?

Coverage

– How can coverage information and constraint satisfaction results be used in in test.
– What are meaningful and useful coverage metrics, and what is the relation of design quality to these metrics?
– How do complexity measures of design relate to coverage?
– Can we do automatic coverage-directed stimulus generation?
– Can data mining be used for effective coverage data analysis?

Example: Experiment with different coverage metrics (e.g., various code-coverage metrics, functional coverage models at various levels, etc.) to explore the correlation between the different approaches and types of coverage models with the probability of revealing specific types of bugs in the design.

Example: Develop learning-based algorithms that will allow a coverage-directed generation process with feedback. More specifically, learn how to use coverage data from a given simulation "cycle" to modify the input parameters for the stimulus generator so that the next simulation "cycle" will produce better coverage results.

Verification Core Technologies

– How can automatic methods for problem reduction, such as symmetry, counters, etc, best be employed?
– What are the most promising verification techniques combining BDDs, SAT, ATPG-based methods, and new approaches?
– How can reachability analysis be improved?
– How can verification methods and algorithms be targeted to the properties to be verified?
– What are formal debugging aids that support fast error localization?
– Are there promising bounded model checking satisfiability solving algorithms?

SRC Task Force in Verification 2002

Dave Reed	AMD	Ken Albin	Motorola
Andreas Kuehlmann	Cadence	Dale Edwards	SRC
Bob Kurshan	Cadence	Justin Harlow	SRC
Tim Leonard	Compaq	Bill Joyner	SRC
Yaron Wolfsthal	IBM	Jim Kukula	Synopsys
Limor Fix	Intel	Carl Pixley	Synopsys
John O'Leary	Intel	David Yeh	
Carl-Johan Seger	Intel		